Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: L15 and programmable

Display: 10 **Documents in Display Format:** KWIC **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Thursday, January 13, 2005 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L16	L15 and programmable	6	L16
L15	(thread$ adj3 select$ adj3 process$)	59	L15
L14	L9 and (select$ adj3 thread$)	0	L14
L13	L9 and (select$ adj3 process$)	1	L13
L12	(thread$ with select$ with process$) and (programmable adj2 engine$)	0	L12
L11	(thread$ with select$ with process$)	891	L11
L10	L9 and (thread$ with select$ with process$)	0	L10
L9	(multiple adj2 threads) and (programmable adj2 engine$)	7	L9
L8	(multiple adj2 threads) and (engine$ with programmable)	14	L8
L7	(threads) and (engine$ with programmable)	117	L7
L6	(multiple with threads with engine$ with programmable)	0	L6
L5	L1 and (programmable)	0	L5
L4	L1 and (engine$ with programmable)	0	L4
L3	L1 and (threads with engine$ with programmable)	0	L3
L2	L1 and (multiple with threads with engine$ with programmable)	0	L2
L1	6604125.pn.	1	L1

h e b b g e e e e f ff e ch

END OF SEARCH HISTORY

h e b b g e e e e f ff e ch

US006529983B1

(12) **United States Patent**
Marshall et al.

(10) **Patent No.: US 6,529,983 B1**
(45) **Date of Patent: Mar. 4, 2003**

(54) **GROUP AND VIRTUAL LOCKING MECHANISM FOR INTER PROCESSOR SYNCHRONIZATION**

(75) Inventors: **John William Marshall**, Cary, NC (US); **Kenneth H. Potter**, Raleigh, NC (US)

(73) Assignee: **Cisco Technology, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/432,464**

(22) Filed: **Nov. 3, 1999**

(51) **Int. Cl.**[7] **G06F 12/00**
(52) **U.S. Cl.** **710/200**
(58) **Field of Search** 710/200, 108; 709/213, 214, 215, 248, 200; 707/8; 711/100, 147, 150, 153, 163, 168

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,589,092	A	* 5/1986	Matick	711/207
4,590,554	A	5/1986	Glazer et al.	364/200
4,598,400	A	7/1986	Hillis	370/60
4,709,327	A	11/1987	Hillis et al.	364/200
4,773,038	A	9/1988	Hillis et al.	364/900
4,791,641	A	12/1988	Hillis	371/38
4,805,091	A	2/1989	Thiel et al.	364/200
4,809,202	A	2/1989	Wolfram	364/578
4,870,568	A	9/1989	Kahle et al.	364/200
4,922,486	A	5/1990	Lidinsky et al.	370/60
4,949,239	A	8/1990	Gillett, Jr. et al.	364/200
4,965,717	A	10/1990	Cutts, Jr. et al.	364/200
4,993,028	A	2/1991	Hillis	371/39.1
5,070,446	A	12/1991	Salem	395/500
5,093,801	A	3/1992	White et al.	364/726
5,111,198	A	5/1992	Kuszmaul	340/825.52
5,113,510	A	5/1992	Hillis	395/425
5,117,420	A	5/1992	Hillis et al.	370/60
5,124,981	A	6/1992	Golding	
5,129,077	A	7/1992	Hillis	395/500
5,148,547	A	9/1992	Kahle et al.	395/800
5,151,996	A	9/1992	Hillis	395/800
5,157,663	A	10/1992	Major et al.	371/9.1
5,163,149	A	11/1992	Brantley et al.	
5,175,852	A	* 12/1992	Johnson et al.	707/8
5,175,865	A	12/1992	Hillis	395/800
5,212,773	A	5/1993	Hillis	395/200
5,222,216	A	6/1993	Parish et al.	395/275
5,222,237	A	6/1993	Hillis	395/650
5,247,613	A	9/1993	Bromley	395/200
5,247,694	A	9/1993	Dahl	395/800
5,255,291	A	10/1993	Holden et al.	375/111
5,261,105	A	11/1993	Pottet et al.	395/725
5,265,207	A	11/1993	Zak et al.	395/200
5,274,631	A	12/1993	Bhardwaj	370/60

(List continued on next page.)

OTHER PUBLICATIONS

Computer Architecture *The Anatomy of Modern Processors*, Pipeline Hazards, world wide web page http://ciips.ee.uwa.edu.au/~morris/CA406/pipe_hazard.html, 1999, pp. 1–6.

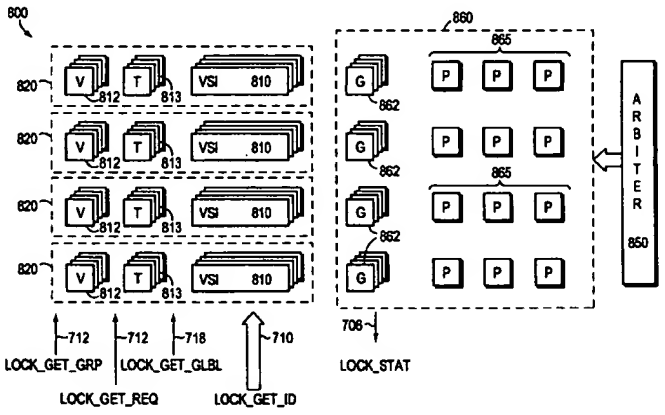
U.S. patent application Ser. No. 09/106,478, Kerr et al., filed Jun. 29, 1998.

Primary Examiner—Xuan M. Thai
(74) *Attorney, Agent, or Firm*—Cesari and McKenna, LLP

(57) **ABSTRACT**

A group and virtual locking mechanism (GVLM) addresses two classes of synchronization present in a system having resources that are shared by a plurality of processors: (1) synchronization of the multi-access shared resources; and (2) simultaneous requests for the shared resources. The system is a programmable processing engine comprising an array of processor complex elements, each having a microcontroller processor. The processor complexes are preferably arrayed as rows and columns. Broadly stated, the novel GVLM comprises a lock controller function associated with each column of processor complexes and lock instructions executed by the processors that manipulate the lock controller to create a tightly integrated arrangement for issuing lock requests to the shared resources.

20 Claims, 7 Drawing Sheets



Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L19: Entry 5 of 6 File: USPT Apr 24, 2001

DOCUMENT-IDENTIFIER: US 6223274 B1
TITLE: Power-and speed-efficient data storage/transfer architecture models and design methodologies for programmable or reusable multi-media processors

Abstract Text (1):
A programmable processing engine and a method of operating the same is described, the processing engine including a customized processor, a flexible processor and a data store commonly sharable between the two processors. The customized processor normally executes a sequence of a plurality of pre-customized routines, usually for which it has been optimized. To provide some flexibility for design changes and optimizations, a controller for monitoring the customized processor during execution of routines is provided to select one of a set of pre-customized processing interruption points and for switching context from the customized processor to the flexible processor at the interruption point. The customized processor can then be switched off and the flexible processor carries out a modified routine. By using sharable a data store, the context switch can be chosen at a time when all relevant data is in the sharable data store. This means that the flexible processor can pick up the modified processing cleanly. After the modified processing the flexible processor writes back new data into the data store and the customized processor can continue processing either where it left off or may skip a certain number of cycles as instructed by the flexible processor, before beginning processing of the new data.

Brief Summary Text (15):
Many of such architectures have been proposed for video and image processing. Power management and power reduction for these processors is hardly tackled in literature but it is recognized as a growing problem in the industry (at least at the "customer" side). Several recent commercial multi-media oriented processors have been marketed or announced: TI-C80 and recently C60, Philips-TriMedia, Chromatic-Mpact, Nvidia NV1, NEC PIP-RAM. Several other Super-scalar/VLIW processors have been announced with an extended instruction-set for multi-media applications: Intel (MMX), SGI/MPS (MDMX), HP (MAX), DEC (MVI), Sun (VVIS), AMD (MMX), IBM (Java). Also a few more dedicated domain-specific ASIP processors have been proposed, such as the MIPS MPEG2 engine which includes a multi-RISC, several memories and a programmable network.

Brief Summary Text (33):
The present invention includes a programmable processing engine, the processing engine including a customized processor, a flexible processor and a data store commonly sharable between the two processors, the customized processor normally executing a sequence of a plurality of pre-customized routines, comprising: a controller for monitoring the customized processor during execution of a first code portion to flexibly select one of a set of pre-customized processing interruption points in a first routine and for switching context from the customized processor to the flexible processor at the selected interruption point.

Brief Summary Text (34):
The present invention also includes a method of operating a programmable processing engine, the processing engine including a customized processor, a flexible

processor and a data store commonly sharable between the two processors, the customized processor normally executing a sequence of a plurality of pre-customized routines, comprising the steps of: monitoring the customized processor during execution of a first code portion to flexibly select one of a set of pre-customised processing interruption points in a first routine; and switching context from the customized processor to the flexible processor at the selected interruption point. The method preferably includes as a next step executing a second code portion on said flexible processor using at least a part of first data left in the data store by the execution of the first code portion on the customized processor.

Detailed Description Text (63):
All communication between the system processor 32 and the cACU 31 is therefore mainly devoted to resolve only run-time data-dependencies and to explicitly synchronize the evolution of both threads of control (the IP 32 and cACU 31). For the master-master model, this explicit synchronization is limited to the specific points where the original functionality has to be modified (namely at a context switch), as described above with respect to the second embodiment of the present invention, i.e. a context switch takes place whenever a new functionality is added or modified with respect to the initial version of the application, which means that some parts of the original functionality assigned to the cACU 31 are taken over (in a modified form) by the IP 32.

Detailed Description Text (64):
Sometimes, the IP 32 needs to follow the traversal of the cACU 31, specifically when data-dependent conditional branches are being decided locally in the cACU 31 and the context switch needs to happen inside one or several of them. Normally, the traversed paths result in unbalanced conditional threads. Thus, it becomes very difficult to predict when the context switch should happen. To concurrently match the unbalanced evolution several design options are possible all of which are aspects of this embodiment of the present invention:

Detailed Description Text (67):
This architecture allows to define a synchronous co-operative model, mainly constrained by the order of the memory accesses that are subject to custom addressing. Otherwise, both performance and cost (area/power) efficiencies can be very limited. Also, the use of related clocks derived from the same system clock 35 is mainly needed for efficiency in the high-level synchronization between both threads of control (the IP 32 and the cACU 31). In the master-master architectures, it affects the synchronization of both schedules (IP 32 and cACU 31). In the master-slave case, it affects the pipelining of the operations of both blocks. In case of related clocks suffering clock-skew problems, it is possible to use a low-level asynchronous protocol for the communication of the data-dependencies, independently of the model chosen.

Detailed Description Text (68):
A first model allows a master-master concurrent evolution of the two threads of control (one for cACU 31 and one for the IP 32). This model is a specific case of the custom block described in a more generic way above in the first embodiment of the present invention and refined for a cACU 31 as an individual embodiment of the present invention. The synchronization in this case is implicit by estimating/controlling, but at compile time, the elapsed time between the memory-transfers subject to custom addressing. Typically, for data-transfer intensive applications, the memory accesses to the slower memories dominate the overall system timing. Thus, one possibility is to concentrate on the memory-transfer and assume that the compiler will schedule the transfers such that they are kept in the same order as in the original algorithm. Therefore, by controlling the compiler (as it is the case for some ASIP's) it is possible to predict when the memory-transfer is taking place. Another possibility to gather this timing information is by analyzing the scheduling output of the compiler. Note, that there is no need to have all details of the scheduling but just these related to the memory access

operations which are subject to custom addressing. In both cases, no control-flow decisions shall be left at run-time (e.g., hardware-cache, interrupts, etc.) to prevent modifications on the scheduling after compilation.

Detailed Description Text (96):
Three instructions are needed to perform the complete normal and modified operating modes plus the already existing load/store operation. Two of them, data-ready and skip-acu (#states) are common to both embodiments: master-master and master-slave. However, the third instruction differs slightly in semantics. In both models the third instruction "means" that an action in the cACU 31 should be started. In the master-master model the "start" action triggers or continues the control thread of the cACU 31 (start-acu). In the master-slave model, it just triggers one super-state transition of the corresponding control-thread.

Detailed Description Text (124):
The implementation of the third embodiment comes partly at the price of reduced flexibility compared to general-purpose RISCs but especially needs a heavy investment in new processor architecture design and compiler technology. The potential savings will be even larger than in the previously described embodiments, however. Recently, some companies have been proposing very domain-specific processors for a limited class of applications. An example is the programmable MIPS MPEG2 engine. These processors still have a limited scope and the data transfer and storage bottle-neck have not been really solved, especially in terms of power. The performance of the architecture model of the third embodiment can be summarized as:

Current US Original Classification (1):
712/34

Current US Cross Reference Classification (1):
712/40

CLAIMS:

1. A programmable processing engine, the processing engine including a customized processor, a flexible processor and a data store commonly sharable between the two processors, the customized processor normally executing a sequence of a plurality of pre-customized routines the programmable processing engine, comprising:

a controller for monitoring the customized processor during execution of a first code portion to select one of a set of pre-customized processing interruption points in a first routine and for switching context from the customized processor to the flexible processor at the interruption point.

3. The processing engine of claim 1, wherein the data store is data storage shared commonly by both the customized and the programmable processor.

5. The processing engine of claim 1, wherein the programmable processor is an application specific instruction set processor.

7. The processing engine of claim 1, wherein the programmable processor includes a counter means for determining the timing of the context switch.

8. The processing engine of claim 1, wherein the customized processor is adapted supply information to the programmable processor sufficient to determine the timing of the context switch.

9. The processing engine of claim 8, wherein the programmable processor is adapted to monitor the branch evolution in the customized processor.

10. The processing engine of claim 8, wherein the programmable processor has a register and the customized processor is adapted to transmit information relating to the status of routines running on the custom processor for storage in said registers.

19. A method of operating a programmable processing engine, the processing engine including a customized processor, a flexible processor and a data store commonly sharable between the two processors, the customized processor normally executing a sequence of a plurality of pre-customized routines the method, comprising:

monitoring the customized processor during execution of a first code portion to select one of a set of pre-customized processing interruption points in the first routine; and

switching context from the customized processor to the flexible processor at the interruption point.

Previous Doc Next Doc Go to Doc#

US006223274B1

(12) **United States Patent**
Catthoor et al.

(10) **Patent No.: US 6,223,274 B1**
(45) **Date of Patent: Apr. 24, 2001**

(54) **POWER-AND SPEED-EFFICIENT DATA STORAGE/TRANSFER ARCHITECTURE MODELS AND DESIGN METHODOLOGIES FOR PROGRAMMABLE OR REUSABLE MULTI-MEDIA PROCESSORS**

(75) Inventors: **Francky Catthoor**, Temse; **Miguel Miranda**, Heverlee; **Stefan Janssens**, Lennik; **Hugo De Man**, Kessel-Lo, all of (BE)

(73) Assignee: **Interuniversitair Micro-Elecktronica Centrum (IMEC)**, Leuven (BE)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/196,645**

(22) Filed: **Nov. 19, 1998**

Related U.S. Application Data

(60) Provisional application No. 60/066,163, filed on Nov. 19, 1997.

(51) **Int. Cl.**[7] **G06F 9/44**

(52) **U.S. Cl.** **712/34**; 712/40; 717/4; 709/108

(58) **Field of Search** 712/34, 35, 36, 712/40, 43; 709/108; 717/4

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,732,275	*	3/1998	Kullick	717/11
5,784,611	*	7/1998	Thantrakul	713/1
5,959,689	*	9/1999	DeLange	348/571
5,974,454	*	10/1999	Apfel	709/221
6,009,507	*	12/1999	Brooks	712/28
6,061,711	*	5/2000	Song	709/108
6,141,675	*	10/2000	Slavenburg	708/706

OTHER PUBLICATIONS

Brodersen, *Proc. IEEE Int. Solid-State Circ. Conf.*, San Francisco, CA, pp. 32–36, Feb. 1997, 9 pages "The Network Computer and its Future."

Catthoor, et al., *IEEE Workshop on VLSI Signal Processing*, La Jolla, CA, pp. 178–187, Oct. 1994, "Global Communication and Memory Optimizing Transformation for Low Power Signal Processing Systems."

Chatterjee, *Proc. IEEE Int. Solid State Circ. Conf.*, San Francisco, CA, pp. 26–30, Feb. 1995, "Gigachips: Deliver Affordable Digital Multimedia for Work and Play Via Broadband Network and Set-Top Box."

(List continued on next page.)

Primary Examiner—Eric Coleman
(74) *Attorney, Agent, or Firm*—Knobbe, Martens, Olson & bear, LLP

(57) **ABSTRACT**

A programmable processing engine and a method of operating the same is described, the processing engine including a customized processor, a flexible processor and a data store commonly sharable between the two processors. The customized processor normally executes a sequence of a plurality of pre-customized routines, usually for which it has been optimized. To provide some flexibility for design changes and optimizations, a controller for monitoring the customized processor during execution of routines is provided to select one of a set of pre-customized processing interruption points and for switching context from the customized processor to the flexible processor at the interruption point. The customized processor can then be switched off and the flexible processor carries out a modified routine. By using sharable a data store, the context switch can be chosen at a time when all relevant data is in the sharable data store. This means that the flexible processor can pick up the modified processing cleanly. After the modified processing the flexible processor writes back new data into the data store and the customized processor can continue processing either where it left off or may skip a certain number of cycles as instructed by the flexible processor, before beginning processing of the new data.

30 Claims, 23 Drawing Sheets